Matrix techniques for faster routing of affine permutations on a mesh interconnection network

J.F. Sibeyn

RUU-CS-90-19
April 1990

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

ISSN:0924-3275

Matrix Techniques for Faster Routing of Affine Permutations on a Mesh Interconnection Network

Jop F. Sibeyn*
Department of Computer Science, University of Utrecht
P.O. Box 80.089, 3508 TB Utrecht , the Netherlands
Email: jopsi@ruuinf.cs.ruu.nl

February 28, 1991

Abstract

We study the problem of routing affine permutations on a SIMD MESH-connected network without wrap-around connections. For a $\sqrt{N} \times \sqrt{N}$ MESH affine permutations can be described by an invertible $\log N \times \log N$ matrix A and a translation vector $\bar{b}$. Thus if the bit-row of the index of a processing-unit is $\bar{x}$ then the bit-row of its destination is $\bar{y} = A \cdot \bar{x} + \bar{b}$. Previously, [9], we performed the routing by a sequence of invertible bit complementations. Those bit complementations were found by using an LUL-decomposition of the matrix A. Refining this approach we found an algorithm using $6 \cdot \sqrt{N} - 6$ routing steps at most and $4 \cdot \sqrt{N} + \mathcal{O}(1)$ on the average. We will improve on this result by using a TUL-decomposition of A where T consists of a number of bit interchanges. We are able to intermix the permutations of T with those of the UL-part. In this way we get an algorithm which needs $4 \cdot \sqrt{N} - 4$ routing steps at most. The permutation is performed by a sequence of selective bit complementations but they are no longer invertible and we accept that two data-sets reside in the same PU during the routing. Our algorithm is optimal for some affine permutations and on the average the number of routing steps is only $\mathcal{O}(1)$ from a lower bound (cf. [9]).

1 Introduction

1.1 Machine model

We are working on a SIMD array processor consisting of $N = 2^n$ (n even) processing units (PUs) organized in a square grid without wrap-around connections: The $\sqrt{N} \times \sqrt{N}$ MESH. PUs are numbered according to the shuffled-row-major scheme (this can be generalized, cf. section 4). The PU numbers are thought of as binary n-vectors (denoted by an italic lower case letter with a bar over it e.g. $\bar{x}$).

*The work of the author was financially supported by the Foundation for Computer Science (SION) of the Netherlands Organization for Scientific Research (NWO). This research was partially supported by the ESPRIT II Basic Research Actions Project of the EC under contract No. 3075 (Project ALCOM).

1.2 Context and results

One of the fundamental problems in parallel computation is the routing problem on an N processor network: Data from PU_i must be routed to the destination processor $\mathrm{PU}_{d(i)}$ and this should be done for all i $(0 \leq i < N)$ simultaneously. In many applications d is a permutation. A trivial way of routing permutations is by sorting on an appended destination field. Sorting algorithms for the MESH are well-known, [6, 10, 3], the most efficient one requiring $(8 \cdot \sqrt{N} + \mathcal{O}(N^{1/3} \cdot n))$ routing steps (rss). Many recent articles on routing and sorting on the MESH, [8, 2, 4], assume a MIMD machine and often have average-case performance. Although these results cannot readily be compared to our results for SIMD machines, they show a vivid interest in the subject. In case of special permutations more efficient routing schemes are possible: For bit-oriented permutations Nassimi & Sahni [5] gave an optimal algorithm needing $4 \cdot \sqrt{N} - 4$ rss at most. If the permutation d is given by $d(\overline{x}) = A \cdot \overline{x} + \overline{b}$, where A is an invertible $n \times n$ 0-1 matrix and $\overline{b}$ is an n-vector, then we call it affine. The class of affine permutations (aps) contains the class of bit-oriented permutations. Aps will be denoted by their defining matrix-vector pair, e.g. $(A, \overline{b})$. In this paper we give a $4 \cdot \sqrt{N} - 4$ rss algorithm for aps. It is an improvement of our previous work [9] which was based on the algorithm of Pease [7] for routing aps on a hypercube network. There we needed $6 \cdot \sqrt{N} - 4$ rss at most and $4 \cdot \sqrt{N} + \mathcal{O}(1)$ rss on the average for routing aps. Thus the main gain is the lower worst-case upper bound.

1.3 Approach

The basic routing operation we use is the (selective) bit-complementation (bc). The i^{th} bc, bc_i, is the permutation

$$\overline{x} = (x_{n-1}, \ldots, x_i, \ldots, x_0) \overset{\mathrm{bc}_i}{\longmapsto} (x_{n-1}, \ldots, x_i + 1, \ldots, x_0) .$$

On a MESH bc_i requires $2 \cdot 2^{\lfloor i/2 \rfloor}$ rss (left/right shifts if i even, up/down shifts if i odd). In a selective bc only part of the $\overline{x}$ vectors participate in the mapping (for the remaining $\overline{x}$ we have $\overline{x} \mapsto \overline{x}$). Generally a selective bc is not a permutation. If it is, we call it invertible. Especially, this is the case if the selective bc is an ap itself and can be represented by $(M, \overline{v})$ for some invertible matrix M and vector $\overline{v}$ which are trivial outside row_i for some i. If BC_i performs the routing of bc_i and every PU has registers *olddata* and *newdata* (initially the *newdata* are put to some dummy value), then selective bcs can be implemented by

Proc BCRoute(i);
1. **for** all $\overline{x}$: Determine the value of $selected_{\overline{x}}$;
2. **for** all $\overline{x}$: **if** $selected_{\overline{x}}$ **then** exchange $olddata_{\overline{x}}$ and $newdata_{\overline{x}}$ **fi** ;
3. **for** all $\overline{x}$: $\mathrm{BC}(i, newdata_{\overline{x}})$.

Lemma 1 *A selective bc of the i^{th} bit can be routed with $2 \cdot 2^{\lfloor i/2 \rfloor}$ rss.*

We will find a permutation matrix B such that the matrix of A with respect to the basis changed with B, $B \cdot A \cdot B^{-1}$, can be written as

$$B \cdot A \cdot B^{-1} = T \cdot U \cdot L,$$

where U is upper triangular, L lower triangular and T a product of bit interchanges. In [9] we gave an easy algorithm for routing the $U \cdot L$ part of this product with $4 \cdot \sqrt{N} - 4$

rss, by executing every bc at most once. Subsequently routing T with the algorithm of Nassimi and Sahni [5] gives an algorithm which needs at most $8 \cdot \sqrt{N} - 8$ rss. However, we can combine the routings of T with those of $U \cdot L$. Thus we find an algorithm which routes aps with $4 \cdot \sqrt{N} - 4$ rss in total. To route these combinations we need non-invertible bcs. A consequence of this is that two data-sets may reside in one PU. We proved in [9] that in case invertible bcs are the only allowed basic routing operations, at least $6 \cdot \sqrt{N} - 6$ are needed for a specific example. In the remainder we will fix the sequence of the n bcs and the values of $selected_{\overline{x}}$.

1.4 Notation

$$
\begin{aligned}
\overline{e_i} &= (0 \ldots 0\, 1_i\, 0 \ldots 0), \text{ the } i^{th} \text{ basis-vector}, \\
I &= n \times n \text{ identity matrix}, \\
x_i &= \overline{e_i} \cdot \overline{x}, \text{ the } i^{th} \text{ element of } \overline{x}, \\
\overline{x_i} &= \overline{x} \cdot \overline{e_i}, \text{ the } i^{th} \text{ element vector of } \overline{x}, \\
\overline{A}_{i,-} &= \overline{e_i} \cdot A, \text{ the } i^{th} \text{ row of } A, \\
A_{ij} &= \overline{e_i} \cdot A \cdot \overline{e}_j, \text{ the element at position } ij \text{ of } A, \\
Ex^{(i\,j)} &= \text{the matrix giving the interchange of } bit_i \text{ and } bit_j, \\
EC^{(k\,l)} &= Ex^{(k\,k+1)} \cdot \ldots \cdot Ex^{(l-1\,l)}, \text{ the elementary cycle of } bit_k \text{ to } bit_l \ (k \leq l).
\end{aligned}
$$

The elementary cycle $EC^{(k\,l)}$ $(k \leq l)$ can be represented by

$$EC^{(k\,l)} = (n-1 \mapsto n-1, \ldots, l+1 \mapsto l+1, l \mapsto k, l-1 \mapsto l, \ldots, k \mapsto k+1, k-1 \mapsto k-1, \ldots, 0 \mapsto 0).$$

As convention (non-standard) on the indices of vectors and matrices we use

$$\overline{x} = \begin{pmatrix} x_{n-1} \\ \vdots \\ x_0 \end{pmatrix}; \; A = \begin{pmatrix} A_{n-1\,n-1} & \ldots & A_{n-1\,0} \\ \vdots & & \vdots \\ A_{0\,n-1} & \ldots & A_{0\,0} \end{pmatrix}.$$

Under this convention elementary cycles have the form

$$EC^{(k\,l)} = \left(\begin{array}{c|cccccc|c}
\begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix} & & & & & & & \\
\hline
 & 0 & 1 & 0 & \ldots & 0 & 0 & \\
 & 0 & 0 & 1 & & & 0 & \\
 & \vdots & & & \ddots & & \vdots & \\
 & 0 & & & & 1 & 0 & \\
 & 0 & 0 & & & 0 & 1 & \\
 & 1 & 0 & 0 & \ldots & 0 & 0 & \\
\hline
 & & & & & & & \begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix}
\end{array}\right).$$

2 Decomposition of the affine permutation

With the notation introduced in section 1.4 we can express more clearly what our decomposition will be: We will find a basis-change given by B, such that we get

$$B \cdot A \cdot B^{-1} = EI^{(0)} \cdot \ldots \cdot EI^{(n-1)} \cdot VL^{(n-1)} \cdot \ldots \cdot VL^{(0)}. \tag{1}$$

Here $EI^{(i)}$ equals $Ex^{(i\,i+1)}$ or I and $VL^{(i)}$ is invertible and trivial outside row_i. The algorithm proceeds as follows:

1. $B := I$;
2. **for** $i := 0$ **to** $n-1$ **do**
   a. **if** $A_{i\,i} = A_{i+1\,i} = 0$
      then select $j > i+1$ such that $A_{j\,i} = 1$;
      $B := Ex^{(i+1\,j)} \cdot B$;
      $A := Ex^{(i+1\,j)} \cdot A \cdot Ex^{(i+1\,j)}$;
      for $k := 0$ **to** $i-1$ **do** $VL^{(k)} := Ex^{(i+1\,j)} \cdot VL^{(k)} \cdot Ex^{(i+1\,j)}$ **od fi** ;
   b. **if** $A_{i\,i} = 0$
      then $EI^{(i)} := Ex^{(i\,i+1)}$;
      $A := Ex^{(i\,i+1)} \cdot A$
      else $EI^{(i)} := I$ **fi** ;
   c. $VL^{(i)} := I; \overline{VL^{(i)}_{i,-}} := \overline{A}_{i,-}$;
      $A := A \cdot VL^{(i)}$ **od** ;

The following invariant property holds at the end of pass i, $-1 \leq i \leq n-1$, of the loop:

$$B \cdot A \cdot B^{-1} = EI^{(0)} \cdot \ldots \cdot EI^{(i)} \cdot A^{(i)} \cdot VL^{(i)} \cdot \ldots \cdot VL^{(0)}, \tag{2}$$

with B a permutation matrix, $EI^{(i)}$, $VL^{(i)}$ as indicated above and $A^{(i)}$ invertible and trivial in $row_0, \ldots, row_i$, the A we find during the algorithm. For $i = -1$ (2) is satisfied. Assume (2) holds at the end of pass $i-1$. Because $A^{(i-1)}$ is invertible there is a $j \geq i$ such that $A^{(i-1)}_{j\,i} = 1$. If it is necessary to make $A^{(i-1)}_{i+1\,i} = 1$, then B is changed. This may induce changes on the $VL^{(j)}$ as well, but their properties are preserved. $Ex^{(i+1\,j)}$ commutes with $EI^{(k)}$ for $k < j$. If at the start of step b. $A^{(i-1)}_{i\,i} = 0$, then this is corrected by exchanging row_i and row_{i+1}. Because ${VL^{(i)}}^{-1} = VL^{(i)}$ we have $(A^{(i-1)} \cdot VL^{(i)}) \cdot VL^{(i)}$. Putting $A^{(i)} = A^{(i-1)} \cdot VL^{(i)}$, it is easy to check that $\overline{A^{(i)}} = \overline{e}_i$. So (2) also holds at the end of pass i. The algorithm as given is correct but very inefficient. E.g., $Ex^{(i+1\,j)} \cdot VL^{(k)} \cdot Ex^{(i+1\,j)}$ can be calculated in $\mathcal{O}(1)$. Neither there is any need to store all trivial matrix rows occurring. Performing this kind of optimalizations we get

Lemma 2 *A decomposition as in (1) can be constructed in $\mathcal{O}(n^3)$ time with $\mathcal{O}(n^2)$ space.*

The decomposition of (1) does not look like a TUL-decomposition. It is, however, closely related to a TUL-decomposition. If we construct a TUL-decomposition of A analogously to the algorithm given above, then $T = EI^{(0)} \cdot \ldots \cdot EI^{(n-1)}$ and, if we put $V = U^{-1}$, then $VL^{(i)} = (I$ with row_i replaced by $\overline{V}_{i,-}) \cdot (I$ with row_i replaced by $\overline{L}_{i,-})$. So (1) could also be obtained from a TUL-decomposition.

Define for any vector $\overline{x}$, matrix A and invertible matrix B $\overline{x'} = B \cdot \overline{x}, A' = B \cdot A \cdot B^{-1}$. For a PU with number $\overline{x}$ we call $\overline{x'}$ its index. A processor with index $\overline{x'}$ will be denoted by

$PU^{\overline{x'}}$. Of course $PU^{\overline{x'}} = PU_{B^{-1}\cdot\overline{x'}}$. An ap $(A, \overline{b})$ is routed by routing $(A', \overline{b'})$ with respect to the indices, i.e. by sending the data from $PU^{\overline{x'}}$ to $PU^{\overline{y'}}$ with $\overline{y'} = A' \cdot \overline{x'} + \overline{b'}$. The B in (1) is bit-oriented. For this case we proved in [9] that routing with respect to the indices is just as easy as routing with respect to the numbers (instead of calling bc_i one should call bc_j, with j such that $B^{-1} \cdot \overline{e_i} = \overline{e_j}$). Therefore, without loss of generality we assume in the following that $B = I$. Now, taking together the consecutive $Ex^{(j\,j+1)}$ and using the definition of $EC^{(k\,l)}$, (1) can be reduced to

$$A = EC^{(k_s l_s)} \cdot \ldots \cdot EC^{(k_0 l_0)} \cdot VL^{(n-1)} \cdot \ldots \cdot VL^{(0)}, \tag{3}$$

with $0 \leq k_i < l_i < k_{i+1} < l_{i+1} \leq n-1$. We are going to intermix the $EC^{(k_i l_i)}$ with the $VL^{(j)}$. Let $W^{(j)} = (\prod_{\{0<i<s|l_i<j\}} EC^{(k_i l_i)}) \cdot VL_j \cdot (\prod_{\{0<i<s|l_i<j\}} EC^{(k_i l_i)})$, then we can rewrite (3) as

$$\begin{aligned} A \;=\; & (W^{(n-1)} \cdot \ldots \cdot W^{(l_s+1)}) \cdot (EC^{(k_s l_s)} \cdot W^{(l_s)} \cdot \ldots \cdot W^{(k_s)}) \cdot \\ & (W^{(k_s-1)} \cdot \ldots \cdot W^{(l_{s-1}+1)}) \cdot \ldots \cdot \\ & (W^{(k_1-1)} \cdot \ldots \cdot W^{(l_0+1)}) \cdot (EC^{(k_0 l_0)} \cdot W^{(l_0)} \cdot \ldots \cdot W^{(k_0)}) \cdot \\ & (W^{(k_0-1)} \cdot \ldots \cdot W^{(0)}). \end{aligned} \tag{4}$$

It remains to find vectors $\overline{c_i} \in \{\overline{0}, \overline{e_i}\}$ such that $\overline{y} = \overline{b} + A \cdot \overline{x}$ satisfies

$$\begin{aligned} \overline{y} \;=\; & (\overline{c_{n-1}} + W^{(n-1)} \cdot \ldots \cdot (\overline{c_{l_s+1}} + W^{(l_s+1)} \cdot (EC^{(k_s l_s)} \cdot (\overline{c_{l_s}} + W^{(l_s)} \cdot \ldots \cdot (\overline{c_{k_s}} + W^{(k_s)} \cdot \\ & (\overline{c_{k_s-1}} + W^{(k_s-1)} \cdot \ldots \cdot (\overline{c_{l_{s-1}+1}} + W^{(l_{s-1}+1)} \cdot \ldots \cdot \\ & (\overline{c_{k_1-1}} + W^{(k_1-1)} \cdot \ldots \cdot (\overline{c_{l_0+1}} + W^{(l_0+1)} \cdot (EC^{(k_0 l_0)} \cdot (\overline{c_{l_0}} + W^{(l_0)} \cdot \ldots \cdot (\overline{c_{k_0}} + W^{(k_0)} \cdot \\ & (\overline{c_{k_0-1}} + W^{(k_0-1)} \cdot \ldots \cdot (\overline{c_0} + W^{(0)} \cdot \overline{x})) \ldots). \end{aligned} \tag{5}$$

For invertible A we have $\overline{b} + A \cdot \overline{x} = A \cdot (A^{-1} \cdot \overline{b} + \overline{x})$, furthermore ${W^{(i)}}^{-1} = W^{(i)}$ and $W^{(i)} \cdot \overline{e_i} = \overline{e_i}$. These relations are used in the following algorithm which calculates the vectors $\overline{c_i}$:

1. Construct a decomposition of A as in (4);
2. **for** $j := n-1$ **to** 0 **do**
   a. **if** $j = l_i$ for some $s \geq i \geq 0$ **then** $\overline{b} := {EC^{(k_i l_i)}}^{-1} \cdot \overline{b}$ **fi** ;
   b. $\overline{b} := W^{(j)} \cdot \overline{b}$; $c_j := b_j$; $b_j := 0$ **od** ;

Step 1 can be carried out with aid of the algorithm of section 2 in $\mathcal{O}(n^3)$ time (c.f. lemma 2). During step 2 we always have $b_i = 0 \,\forall i > j$. Step 2.a and step 2.b can be implemented such that they only cost $\mathcal{O}(l_i - k_i)$ and $\mathcal{O}(j+1)$ time, respectively. Thus step 2 requires $\mathcal{O}(n^2)$ time. Concluding

Lemma 3 *In $\mathcal{O}(n^3)$ time we can express $\overline{y} = A \cdot \overline{x} + \overline{b}$ as in (5) with $\mathcal{O}(n^2)$ space.*

We illustrate the process of "bringing $\overline{b}$ into the permutation" with an example:

Example 1 *In the following matrices empty places are zero; at the positions marked "∗" both values may occur.*

$$\overline{y} = \begin{pmatrix} b_2 \\ b_1 \\ b_0 \end{pmatrix} + \begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix} \cdot \overline{x}$$

$$
\begin{aligned}
&= \begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix} \cdot \left(\begin{pmatrix} c_2 \\ b_1 \\ b_0 \end{pmatrix} + \begin{pmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix} \cdot \overline{x} \right) \\
&= \begin{pmatrix} c_2 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{pmatrix} \cdot \left(\begin{pmatrix} 0 \\ b_0 \\ b_1 \end{pmatrix} + \begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix} \cdot \overline{x} \right) \\
&= \begin{pmatrix} c_2 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix} \cdot \left(\begin{pmatrix} 0 \\ c_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix} \cdot \overline{x} \right) \\
&= \begin{pmatrix} c_2 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{pmatrix} \cdot \left(\begin{pmatrix} 0 \\ c_1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix} \cdot \left(\begin{pmatrix} 0 \\ 0 \\ c_0 \end{pmatrix} + \begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix} \cdot \overline{x} \right) \right).
\end{aligned}
$$

3 Routing the permutation

In this section we will give procedures for routing the permutations which constitute (5):

$$
\begin{aligned}
\overline{x}^{(l)} &= \overline{c_l} + W^{(l)} \cdot \ldots \cdot (\overline{c_k} + W^{(k)} \cdot \overline{x}^{(k-1)}), && (6) \\
\overline{x}^{(l)} &= EC^{(kl)} \cdot (\overline{c_l} + W^{(l)} \cdot \ldots \cdot (\overline{c_k} + W^{(k)} \cdot \overline{x}^{(k-1)})). && (7)
\end{aligned}
$$

Because the matrices $W^{(i)}$ and the $\overline{c_i}$ are non-trivial in row_i only these are almost compositions of selective bcs. Permutations as in (6) can be routed using a worked-out form of BCRoute of section 1.3:

Proc BCsRoute($k, l, W^{(k)}, \ldots, W^{(l)}, \overline{c}$);
for $i := k$ **to** l **do**
1. **for** all $\overline{x}$: $selected_{\overline{x}} := (x_i \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{x})$;
2. **for** all $\overline{x}$: **if** $selected_{\overline{x}}$ **then** exchange $olddata_{\overline{x}}$ and $newdata_{\overline{x}}$ **fi** ;
3. **for** all $\overline{x}$: BC($i, newdata_{\overline{x}}$);
4. **for** all $\overline{x}$: **if** $selected_{\overline{x}}$ **then** exchange $olddata_{\overline{x}}$ and $newdata_{\overline{x}}$ **fi od** .

After step 4 of every pass every PU contains exactly one data-set residing in the *olddata*. This is a direct consequence of the bcs being invertible in this case, it can also be expressed by $selected_{\overline{x}} = (x_i \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{x}) = (bc_i(\overline{x})_i \neq c_i + \overline{W^{(i)}}_{i,-} \cdot bc_i(\overline{x})) = selected_{bc_i(\overline{x})}$, where we used $W^{(i)}_{ii} = 1$. ¿From this relation it follows that the permutation consists of pairwise exchanges of data-sets.

The permutation of (7) can only be routed efficiently if we accept non-invertible bcs and accept that some PUs contain temporarily two data-sets. First we give an example:

Example 2 *We consider a permutation of the form of (7) with a cycle of length 3 and $\overline{c} = \overline{0}$:*

$$
\overline{x}^{(2)} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}}_{EC^{(02)}} \cdot \underbrace{\begin{pmatrix} 1 & * & * \\ & 1 & \\ & & 1 \end{pmatrix}}_{W^{(2)}} \cdot \underbrace{\begin{pmatrix} 1 & & \\ * & 1 & * \\ & & 1 \end{pmatrix}}_{W^{(1)}} \cdot \underbrace{\begin{pmatrix} 1 & & \\ & 1 & \\ * & * & 1 \end{pmatrix}}_{W^{(0)}} \cdot \overline{x}^{(-1)} \cdot
$$

$x_1^{(2)}$ is determined by $W^{(0)}$, $x_2^{(2)}$ is determined by $W^{(1)}$, $x_0^{(2)}$ is determined by $W^{(2)}$.

Define $\overline{y}_d^{(i-1)} = \overline{c_{i-1}} + W^{(i-1)} \cdot \ldots \cdot (\overline{c_k} + W^{(k)} \cdot \overline{x}_d^{(k-1)})$. $\overline{y}_d^{(i-1)}$ is the number of the PU a data-set d coming from $\overline{x}_d^{(k-1)}$ would have reached after executing pass $i-1$ of a trivial routing algorithm starting with BCsRoute. From example 2 we see that $x_{d,i+1}^{(l)}$, the final value of x_{i+1} for d, can be expressed in terms of $\overline{y}_d^{(i-1)}$: $x_{d,i+1}^{(l)} := c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{y}_d^{(i-1)}$. This gives an algorithm analogous to BCsRoute for routing the permutation of (7):

Proc ECBCsRoute($k, l, W^{(k)}, \ldots, W^{(l)}, \overline{c}$);
for $i := k$ **to** l **do** { Replace $i+1$ by k if $i = l$. }
1. **for** all $\overline{x}$, data-sets d in $\overline{x}$: calculate $\overline{y}_d^{(i-1)}$;
2. **for** all $\overline{x}$, data-sets d in $\overline{x}$: *selected*$_d$ $:= (x_{i+1} \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{y}_d^{(i-1)})$;
3. **for** all $\overline{x}$, data-sets d in $\overline{x}$: **if** *selected*$_d$ **then** route d to $bc_{i+1}(\overline{x})$ **fi od** .

There are two questions that remain: Can $\overline{y}_d^{(i-1)}$ be calculated from $\overline{x}$; can we guarantee that there is never more than one data-set to be routed from any PU? Assume that at the start of pass i we find in a PU, $\overline{x}$, two data-sets: *olddata*, data that remained at $\overline{x}$ during pass $i-1$ and *newdata*, data that newly arrived. Generally we have $y_{d,i}^{(i-1)} = x_{d,i}^{(k-1)}$. *olddata* was not selected during the routing of bc_i or earlier this gives $x_{olddata,i}^{(k-1)} = x_i$ and thus $y_{olddata,i}^{(i-1)} = x_i$. Furthermore, from step 2 and 3 of ECBCsRoute we see that $y_{olddata,j}^{(i-1)} = x_{j+1}$ for all $i > j \geq k$. With an analogous reasoning for *newdata* we get

$$\begin{aligned}
\overline{y}_{olddata}^{(i-1)} &= (x_{n-1}, \ldots, x_{i+1}, \quad x_i, \quad x_i, x_{i-1}, \ldots, x_{k+1}, x_{k-1}, \ldots, x_0), \\
\overline{y}_{newdata}^{(i-1)} &= (x_{n-1}, \ldots, x_{i+1}, \quad x_i + 1, \quad x_i, x_{i-1}, \ldots, x_{k+1}, x_{k-1}, \ldots, x_0).
\end{aligned}$$

Thus $\overline{y}_{newdata}^{(i-1)} = bc_i(\overline{y}_{olddata}^{(i-1)})$. This gives, use $W_{ii}^{(i)} = 1$, $selected_{olddata} = (x_{i+1} \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{y}_{olddata}^{(i-1)}) \neq (x_{i+1} \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{y}_{newdata}^{(i-1)}) = selected_{newdata}$. This means that either *olddata* or *newdata* should be routed. Starting with data-sets in *olddata* and dummies in *newdata* we thus always remain in a situation as required. Both questions have therefore been settled positively. Now ECBCsRoute can be completed:

Proc ECBCsRoute($k, l, W^{(k)}, \ldots, W^{(l)}, \overline{c}$);
for $i := k$ **to** $l - 1$ **do**
  for all $\overline{x}$: $\overline{y}_{\overline{x}} := (x_{n-1}, \ldots, x_{i+1}, x_i, x_i, x_{i-1}, \ldots, x_{k+1}, x_{k-1}, \ldots, x_0)$;
  for all $\overline{x}$: *selected*$_{\overline{x}}$ $:= (x_{i+1} \neq c_i + \overline{W^{(i)}}_{i,-} \cdot \overline{y}_{\overline{x}})$ { This is *selected*$_{olddata}$ for $\overline{x}$. } ;
  for all $\overline{x}$: **if** *selected*$_{\overline{x}}$ **then** exchange *olddata*$_{\overline{x}}$ and *newdata*$_{\overline{x}}$ **fi** ;
  for all $\overline{x}$: BC($i+1$, *newdata*$_{\overline{x}}$) **od** ;
for all $\overline{x}$: $\overline{y}_{\overline{x}} := (x_{n-1}, \ldots, x_{l+1}, x_l, x_l, x_{l-1}, \ldots, x_{k+1}, x_{k-1}, \ldots, x_0)$;
for all $\overline{x}$: *selected*$_{\overline{x}}$ $:= (x_k \neq c_l + \overline{W^{(l)}}_{l,-} \cdot \overline{y}_{\overline{x}})$;
for all $\overline{x}$: **if** *selected*$_{\overline{x}}$ **then** exchange *olddata*$_{\overline{x}}$ and *newdata*$_{\overline{x}}$ **fi** ;
for all $\overline{x}$: BC(k, *newdata*$_{\overline{x}}$) **od** ;
for all $\overline{x}$: **if** *olddata*$_{\overline{x}}$ = *dummy* **then** exchange *olddata*$_{\overline{x}}$ and *newdata*$_{\overline{x}}$ **fi** .

The last statement is to resolve the non-invertibility. BCsRoute and ECBCsRoute can be combined to give a parallel algorithm for routing permutations as in (5) on a $\sqrt{N} \times \sqrt{N}$ MESH. In this algorithm every bc is used exactly once. With lemma 1 and lemma 3 we get

Theorem 1 *After preprocessing with $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space aps can be routed with $4 \cdot \sqrt{N} - 4$ routing steps.*

We give an example of the data movement occurring in the course of the algorithm:

Example 3 *We consider the permutation* $(EC^{(0\,2)}, \overline{0})$ *on a network with eight PUs:* $PU_{000}, \ldots, PU_{111}$. *The decomposition is trivial in this case with* $W^{(0)} = W^{(1)} = W^{(2)} = I_3$ *and* $\overline{c'} = \overline{0}$, *without need for a change of basis. We just have to route* $T = EC^{(0\,2)}$. *During the execution of the algorithm the PU registers take on the following values:*

$\overline{x}$	000	001	010	011	100	101	110	111	
olddata	d_{000}	d_{001}	d_{010}	d_{011}	d_{100}	d_{101}	d_{110}	d_{111}	*initial situation*
newdata	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	
$\overline{y}$	000	001	010	011	100	101	110	111	$i = 0$
selected	*F*	*T*	*T*	*F*	*F*	*T*	*T*	*F*	
olddata	d_{000}	$\emptyset$	$\emptyset$	d_{011}	d_{100}	$\emptyset$	$\emptyset$	d_{111}	*exchange*
newdata	$\emptyset$	d_{001}	d_{010}	$\emptyset$	$\emptyset$	d_{101}	d_{110}	$\emptyset$	
newdata	d_{010}	$\emptyset$	$\emptyset$	d_{001}	d_{110}	$\emptyset$	$\emptyset$	d_{101}	*routing of* bc_1
$\overline{y}$	000	000	011	011	100	100	111	111	$i = 1$
selected	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*F*	
olddata	d_{000}	$\emptyset$	$\emptyset$	d_{001}	d_{110}	$\emptyset$	$\emptyset$	d_{111}	*exchange*
newdata	d_{010}	$\emptyset$	$\emptyset$	d_{011}	d_{100}	$\emptyset$	$\emptyset$	d_{101}	
newdata	d_{100}	$\emptyset$	$\emptyset$	d_{101}	d_{010}	$\emptyset$	$\emptyset$	d_{011}	*routing of* bc_2
$\overline{y}$	000	000	001	001	110	110	111	111	$i = 2$
selected	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*F*	
olddata	d_{000}	$\emptyset$	$\emptyset$	d_{101}	d_{010}	$\emptyset$	$\emptyset$	d_{111}	*exchange*
newdata	d_{100}	$\emptyset$	$\emptyset$	d_{001}	d_{110}	$\emptyset$	$\emptyset$	d_{011}	
newdata	$\emptyset$	d_{100}	d_{001}	$\emptyset$	$\emptyset$	d_{110}	d_{011}	$\emptyset$	*routing of* bc_0
olddata	d_{000}	d_{100}	d_{001}	d_{101}	d_{010}	d_{110}	d_{011}	d_{111}	*if test*
newdata	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	

4 Optimality, numbering schemes

An example shows that sometimes our algorithm is optimal and sometimes it is not:

Example 4 *Let* $TT = (Ex^{(0\,1)} \cdot Ex^{(2\,3)} \cdot \ldots \cdot Ex^{(n-2\,n-1)}, \overline{0})$; $CC = (I, (1, \ldots, 1))$, *then*

$$\begin{array}{rclrcl} TT(0,1,\ldots,0,1) & = & (1,0,\ldots,1,0), & TT(1,0,\ldots,1,0) & = & (0,1,\ldots,0,1), \\ CC(0,\ldots,0) & = & (1,\ldots,1), & CC(1,\ldots,1) & = & (0,\ldots,0). \end{array}$$

TT cannot be routed by a sequence of bcs in less than $4 \cdot \sqrt{N} - 4$ *rss: One has to carry out every bc. However, TT can be routed in* $2 \cdot \sqrt{N} - 4$ *rss with the algorithm of Nassimi & Sahni [5]. Thus our algorithm does not route TT optimally. On the other hand, a distance argument shows that CC can never be routed with less than* $4 \cdot \sqrt{N} - 4$ *rss on a* $\sqrt{N} \times \sqrt{N}$ *SIMD MESH.*

In [9] we showed that routing over large distances is very common. We proved that on the average an ap needs more than $4 \cdot \sqrt{N} - 16$ rss. This means that our algorithm is $\mathcal{O}(1)$ from optimal on the average. It is possible to save rss if $VL^{(i)} = 0$ for some i. This can be very useful but on the average the improvement is neglectable (if testing time is taken into account it is even a deterioration). Although for many permutations (e.g. TT) our algorithm is not optimal, it is easy to see that, after the "improvement" sketched above, no algorithm using only one-bit operations (selective bcs, invertible or not) uses less rss. Just as we knew at the start of this paper that we needed non-invertible bcs to reach the upper bound of $4 \cdot \sqrt{N} - 4$ rss, we know now that if we want to reach an optimal ap routing algorithm the least we need are two-bit operations. Correct two-bit manipulations are easy to give, the problem is to find a decomposition of the permutation such that every bit is manipulated at most once. Further study is necessary to point out whether and how this generalization of the work of Nassimi & Sahni can be achieved.

If the numbering of the PUs differs from the shuffled-row-major (srm) numbering scheme by some ap $(C, \overline{d})$, then it is easy to express an ap $(A', \overline{b'})$, that should be routed with respect to this numbering, as $(A, \overline{b})$, given with respect to the srm numbering: An $\overline{x}$ given with respect to the srm numbering has modified number $\overline{x'} = C \cdot \overline{x} + \overline{d}$. $\overline{x}$ should be routed to $\overline{y'} = A' \cdot \overline{x'} + \overline{b'} = A' \cdot (C \cdot \overline{x} + \overline{d}) + \overline{b'}$. This $\overline{y'}$ has srm number $\overline{y} = C^{-1} \cdot (\overline{y'} - \overline{d}) = C^{-1} \cdot (A' \cdot (C \cdot \overline{x} + \overline{d}) + \overline{b'} - \overline{d}) = C^{-1} \cdot A' \cdot C \cdot \overline{x} + C^{-1} \cdot (A' \cdot \overline{d} + \overline{b'} - \overline{d})$, so we can take $A = C^{-1} \cdot A' \cdot C$ and $\overline{b} = C^{-1} \cdot (A' \cdot \overline{d} + \overline{b'} - \overline{d})$. This observation allows us to use rather general numbering-schemes. The row-major numbering-scheme is among them, the snake-like numbering-scheme, however, is not.

5 Conclusion

We studied the problem of routing affine permutations on a MESH. We used a decomposition algorithm to rewrite the affine permutation as a composition of affine permutations which where non-trivial in one row only, preceded by some elementary cycles. The routing could be performed now by a sequence of invertible and non-invertible selective bit-complementations. Because every bit-complementation was used at most once a routing time of $4 \cdot \sqrt{N} - 4$ followed.

References

[1] Aho, V. A., J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley PC. 1974.

[2] Krizanc, D., S. Rajasekaran, T. Tsantilas, Optimal algorithms for MESH-connected processor arrays, *Proc. 3th AWOC, Lecture Notes in Comp. Sc. 319*, 1988, pp. 411-422.

[3] Kumar, M. and D.S. Hirschberg. An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes, *IEEE Trans. Comp. C-32 (1983), pp. 254-264.*

[4] Kunde, M., Routing and sorting on MESH-connected arrays, *Proc. 3th AWOC, Lecture Notes in Comp. Sc. 319*, 1988, pp. 423-433.

[5] Nassimi, D. and S. Sahni. An optimal routing algorithm for MESH-connected parallel computers, *Jrnl ACM 27 (1980), pp. 6-29.*

[6] -, Bitonic sort on a MESH-connected parallel computer, *IEEE Trans. Computers, C-27 (1979), pp. 2-7.*

[7] Pease, M.C. The indirect binary n-cube microprocessor array, *IEEE Trans. Comput., C-26 (1977), pp. 458-473.*

[8] Schnorr, C. P., A. Shamir, An optimal sorting algorithm for MESH-connected computers, *Proc. 18th ACM Symp. on Th. of Comp.*, 1986, pp 255-263.

[9] Sibeyn, J.F. Routing affine permutations on a MESH interconnection network by a sequence of bit complementations. *Techn. Rep. Dep. of Comp. Sc. Univ. of Utrecht, to appear.*

[10] Thompson, C.D. and H.T. Kung. Sorting on a MESH-connected parallel computer, *Commun. ACM 20 (1977), pp. 263-271.*